Software Engineering

Lecture 1: Fundamentals

Overview of Lecture

- Introduction to the course
- SE Lifecycle
- Gathering requirements

What is this course about?

- What the stages of software development are
- Uncovering user requirements
- How to analyse and formally document those requirements
- Object Oriented methods, notation and practice
- The structure of a Requirement Specifications document
- Practice
  - Analysing a system
  - Use of a Unified Modelling Language (UML) tool in this process
  - System documentation

Why are you doing this course?

- It is a core course for all of the CS & IT degrees
- What is it that makes this course important?
  - Conceptual models
  - Overview of modelling computer systems
  - Understanding the process that people follow when developing software

Who are the staff?

- Dale Stanbrough
  - Office: 14.10.10a
  - Email: dale@rmit.edu.au
  - Phone: 9925 2652
  - Web: www.cs.rmit.edu.au/~dale
- Consultation:
  - see my website
- Head Tutor
  - Weeks 1-8 Meghna Jaiswal
    - Consultation in 14.9.13

How do I get my questions answered?

- Four options:
  - In class (lecture/tute/lab)
  - In the web discussion forum
  - In email with the head tutor, the demonstrator for the topic or the lecturer
  - At consultation times (see my website)

Lecture topics

- Engineering, Software Engineering, Process
- Requirements
- Introduction to UML / Use Case Analysis
- Analysis of Problems
  - Class diagrams
  - Behavioural modelling
  - State diagrams
- Testing
- Project management
- SQA, Risk
- Process

Practical Sessions

- Each week there are four contact hours in addition to the lectures
  - 2 hour lecture
    - Initial presentation of ideas
    - Working through problems
  - 1 hour tutorial
    - Working through set problems
  - 1 hour laboratory
    - Working with Visual Paradigm to draw UML diagrams
    - Working on your assignments

What to expect

- Assignment results back within 2 weeks (RMIT rule)
- Staff who are responsive
- Staff to turn up on time
  - Mostly staff are great, but...
  - Let the head tutor/me know if they don't
- Lab assistants are there to help, and should be actively asking you questions
  - Don't let them hide in the corner doing their own work!

Assessment

- To pass this course, you must:
  - Pass the practical assessment worth 50%
  - Pass the closed book exam which is 50%
- Your final mark cannot be more than 15% of your exam mark.
  - E.g. if you get full marks for your assignments, yet only 25/50 (50%) on the exam, your final result will be limited to 50 + 15 = 65.
- Late assignments are not accepted.
  - You are required to submit your assignment every day that you work on it.
  - If you are ill/other circumstances intervened, we can take that into account.

Assessment

- Two streams of assignments
  - A1 - A5 - 5 x 4%
    - Work individually creating diagrams
    - Learn the concepts
  - A6, A7 - 2 x 15%
    - Work singly or in pairs
    - Apply your knowledge to examining and documenting (reverse engineering) a real system

Assignments

- Ass1 - Friday 31st July (week 2)
  - Create a list of requirements
- Ass2 - Friday 7th August (week 3)
  - Use case diagrams
- Ass3 - Friday 14th August (week 4)
  - Class/object diagrams
- Ass4 - Friday 21st August (week 5)
  - Class/object diagrams
- Ass5 - Friday 28th August (week 6)
  - Sequence diagrams

Assignments

- Ass6 - Friday 25th August (week 9)
  - Analyse an existing system and produce
    - Use case diagrams
    - Class and object diagrams
- Ass7 - Friday 16th October (week 12)
  - Continue to analyse the same system as Ass6 and produce
    - Activity diagrams
    - Sequence diagrams
    - State diagrams

What is Engineering?

- A body of knowledge used when building things
  - Scheduling
  - Costing
  - Estimating
  - Building
  - Testing
  - Communicating
  - Organising

What is Engineering?

- Engineering involves choices
  - An engineer has to build solutions to problems which are
    - correct
    - cheap
    - adaptable
    - fast
    - reliable
    - safe
    - cost effective
  - An engineer often has to choose when these goals conflict
  - Knowing when choices need to be made and what those choices are, are what distinguishes an engineer from others.

What is Engineering?

- Engineering is about planning
  - Understanding present and future requirements
  - Creating adaptable designs
  - Software should be easily implemented
  - Software should be easily modified

Photo courtesy of The Register (www.register.co.uk)

What is Engineering?

- Engineering is about design
  - Some software is developed without much thought
  - It may start off simply, but due to bad design quickly becomes a mess.
  - Often due to new requirements being inserted without revising the original design.

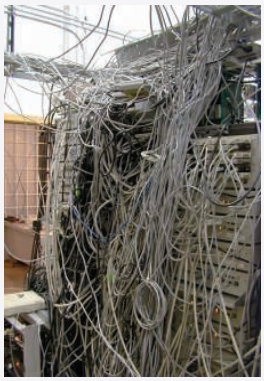


Photo courtesy of The Register (www.register.co.uk)

What is Engineering?

- Engineering is about cost.
  - Is this easy or cheap to maintain?
  - Does your software look like this?
  - How would you go about modifying it for a new purpose?

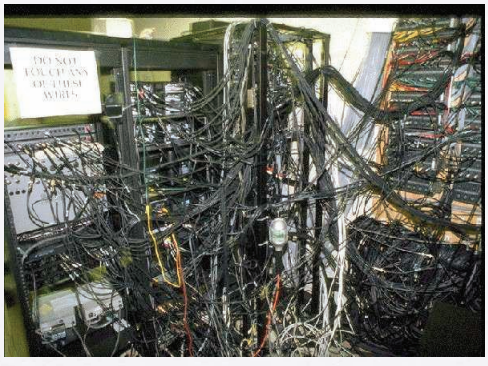


Photo courtesy of The Register (www.register.co.uk)

What is Engineering?

- Is this a well Engineered solution?
- Does it meet the needs of the owners?
- Was it built to budget?
- Is it adaptable?
- Is it safe?
- Does it meet legal requirements?

Photo courtesy of Dale Stanbrough

Components of Software Engineering

Process RUP XP	A framework for the tasks required to build high-quality software.
Notation UML	An agreed, common language used to communicate all aspects of the system, from requirements to final design.
Tools Eclipse Poseidon	Tools to make working through the process less difficult for all involved
Product Your program	The actual software product or system built and put into operation

Engineering involves processes

- Small programs (<10,000 lines) can be implemented easily by one person
- Larger systems need a more systematic approach
- We distinguish between
  - the product (what we are making)
  - the process (how we make it)
- Engineering deals explicitly with process issues

Software Development Life Cycle

- SDLC has several clearly defined phases
  - **Requirements gathering/ elicitation** — Finding out what the user wants
  - **Systems analysis** — Understanding/documenting requirements
  - **Design** — Planning a possible solution
  - **Implementation** — Building a solution
  - **Testing** — Ensuring it meets requirements

Other parts of the life cycle

- Other aspects of developing software
  - Feasibility analysis
    - Is it technically possible to build the system?
    - Does the company have the ability to build it
      - People
      - Skills
      - Finance
      - Time

Other parts of the life cycle

- Other aspects of developing software
  - Project scoping
  - Deals with putting limits on how much you want to create/ clearly specifying what you want to achieve.
    - You may decide to leave network play out of a game.
    - An accounting package may be limited to supporting one database back-end (e.g. Oracle)
    - A music player may only play songs encoded in MP3 format
    - The product may have to be developed with a budget no greater than $400,000
    - The product may have to be created within 12 months

Other parts of the life cycle

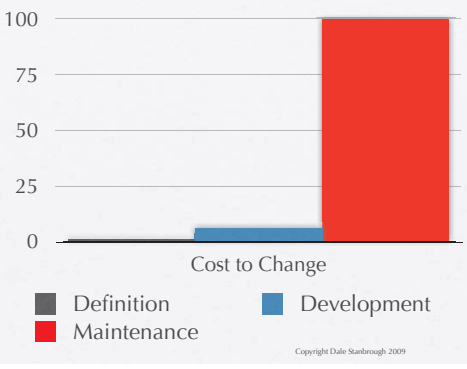
- Other aspects of developing software
  - Project management
    - Estimating development time
    - Ensuring people are used effectively
    - Ensuring good communication and a healthy work environment
    - Managing budgets
  - Training and handover
  - Maintenance
- These issues won't be covered in this course.

Industry status (2004)

- How well are systems built?
  - Total Failure 15%
  - "Challenged" 51%
  - Successful 34%
- In 1994 Successful was only 17%!
- Challenged means…
  - over time, budget, or lacking critical features/requirements

Costs of Correcting Software Errors

- Cost of correcting an error in specifications increases as we move through lifecycle phases
- Ariane 5 specification error cost hundreds of millions of $
- Graph shows the relative costs of fixing an error at different stages of development



Examples development failures

- Denver Baggage Handling System
  - Newly constructed airport to have totally automated baggage handling system
  - System not tested thoroughly - baggage would pile up undetected, and the system would transfer and drop some bags onto one of 4000 telecarts across 34 kms of track.
  - Buggy software caused the system to crash repeatedly. When rebooted the software didn't know which telecarts were loaded.
  - An 11 month delay in opening the airport at $1 Million per day in costs contributed to a total cost overrun of $3Billion.

- A replacement system was installed at a cost of $51 million and was still running 10 years later.
- The project was finally abandoned in June 2005, and had cost over $US700M

Examples development failures

- Arianne 5
  - A successor to the Arianne 4, the more powerful rocket had a different flight profile.
  - Management insisted that the existing Arianne 4 software be reused without being tested against the new flight profile.
  - When launched an overflow triggered an exception, shutting down the guidance computers.
  - The rocket veered off course and was destroyed.
  - $500 million dollars worth of satellites were destroyed.

Why do projects fail?

- "The majority of accidents described in this publication were not caused by some subtle failure of the control system, but by defects that were preventable if a systematic approach had been adopted throughout its design lifecycle.

  Failure to pay attention to detail, particularly during the specification phase of a project, and to properly manage technical issues were the root causes of these accidents."

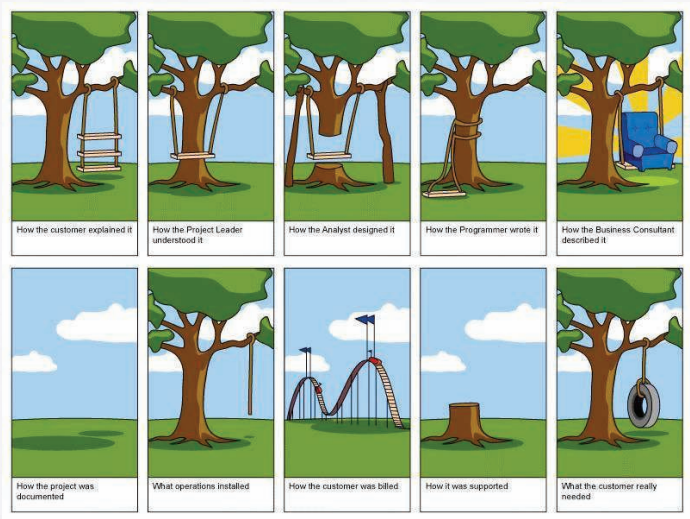
"Out of Control: Why control systems go wrong and how to prevent failure", Sudbury, 1995
HSE Books - Health and Safety Executive, Great Britain. ISBN 0-7176-2192-8

What are the main causes?

- Not having a good understanding of the requirements
- Not knowing how to manage a project
- Trying to develop too much at once

- Can be solved by
  - Iterative development
  - Risk management

Classic view of failure



Process

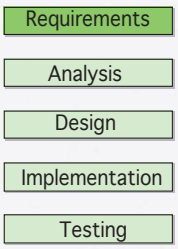
- The algorithm you follow to develop a system
- Many processess...
  - Waterfall - interesting but outdated
  - Rational Unified Process
  - XP
  - Scrum

Waterfall process

- One of the earliest attempts at describing a systematic way of creating software
- Consists of the steps...
  - Requirements gathering
  - Analysis
  - Design
  - Implementation
  - Testing
- It was expected that each stage would be completed before the next was started.

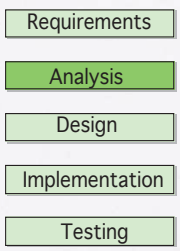
Requirements

- Requirements are things that the user wants the system to do.
- They are grouped into
  - Functional - behaviours we expect the system to have
  - Non Functional - other features to which the system must conform.
- Can be extremely hard to capture
  - Hard to document
  - Hard to get from the user



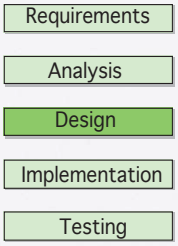
Analysis

- The requirements are analysed and documented in detail - we attempt to have an understanding of what the requirements mean, and search for all possible consequences.
- Sometimes requirements are given to you.
  - You have to understand the terms used
  - Detail the relationships between items
  - Understand the order in which processes must occur
- Sometimes you have to create the requirements
  - Interviewing clients
  - Examining existing systems



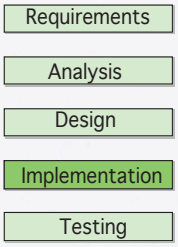
Design

- Analysis is concerned with what the client wants, and what the client understands/cares about.
- Design deals with how the system will be implemented. You might have to choose...
  - Database
  - Implementation Language
  - How to organise the implementation
  - Hardware requirements
- Design stage usually ends with (at least) class descriptions/methods



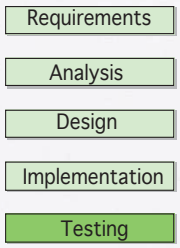
Implementation

- Each class is implemented according to the design document.
- Each class is tested to ensure it matches the class's expected behaviour.
  - Producing correct output given correct input.
  - Meeting timing requirements
- The classes are integrated to produce the final system.

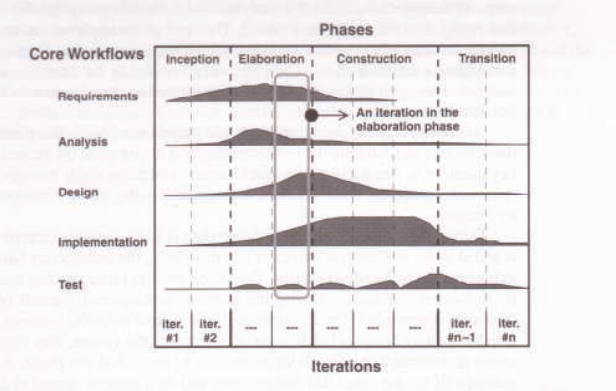


Testing

- The final system is tested to see if it meets the original requirements.
- The components of the system are pieced together, and the interaction between them is tested (integration testing)
- User manuals/procedures are tested
- Backup procedures are tested.



Rational Unified Process



Requirements

- Requirements
  - What are requirements?
  - Classifying the requirements
  - Characteristics of requirements
  - Requirements elicitation
  - Documenting the requirements
  - Requirements analysis and negotiation
  - Requirements validation

Requirements gathering is very hard, and very important

- Before you can finish a project, you have to know what you want to achieve.
  - This may seem obvious, but it is the cause of many failures of development.
- "The hardest single part of building a software system is deciding what to build… No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later".

Frederick P Brooks, "The Mythical Man-Month", Addison-Wesley, 1995, ISBN 0201835959

What are requirements?

- Requirements are a description or statement of a function, feature or condition that a user seeks to have implemented in a system.

- UML has a modelling technique called "Use cases" to help gather requirements.
- These specify all the different ways people interact with a system.
- We examine specific scenarios to test our Use cases.

- How will you know when your project is complete?

Classifying the requirements

- There are 2 broad categories of requirements:
  - Functional Requirements are those that relate directly to the functioning of the system. These are the aspects of the system the client is most likely to recognise.
  - Non-functional requirements cover aspects of the system such as user-interface, performance, quality issues, interfaces to other systems, security etc.

Example functional requirements

- User level facility
  - “The word processor must include a command to check spelling”
- System properties
  - “The system must ensure that personal information is never released without authorisation”
- Specific algorithms
  - “If either hurdle is failed, then the final mark for the subject is the lower of the two percentages”
- Specific constraints
  - “The sensor must be polled 10 times per second”

Example functional requirements

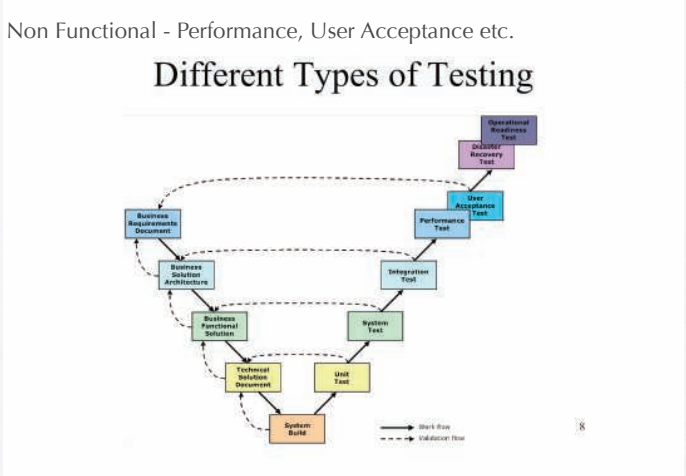
- Legal constraints
  - The legal system may impose constraints on the behaviour of the software by either requiring or banning certain behaviours.
    - "The software must calculate the tax payable and forward it to the Tax Department"
    - "The system must ensure that personal information is kept secure
    - "The system must not allow records to be permanently deleted"

Example non functional requirements

- Constraints on system development
  - "The system must be developed in Java"
- Interface constraints
  - "The user interface must follow the host operating system's guidelines"
- Performance
  - "The system must respond to user requests within 2 seconds for 95% of queries"
- Interfaces to other systems
  - "The system should be capable of integrating information from RSS feeds from websites and displaying them on screen".
- Security
  - "Only authorised users should be able to access the features listed below…"

Testing Functional v.s. Non Functional req.

- Functional testing - This occurs at levels below System test
- Non Functional - Performance, User Acceptance etc.



Characteristics of requirements

- Requirements are expected to be…
  - Verifiable
  - Complete
  - Unambiguous
  - Consistent
  - Modifiable
  - Traceable

Characteristics of requirements

- Verifiable
  - A requirements document often becomes part of the contract between a client and a system developer.
  - It describes what the developer will deliver and be paid for.
  - It is important that you can check if the final product fulfills the requirements
  - Requirements should therefore be verifiable
    - It should be possible to decide if the software meets the requirements.
  - Requirements that contain subjective assessments of quality are always problematic.
  - Example requirements that can't be tested.
    - "The program must be very fast."
      - What is a measure for fast?
      - What aspects should be fast? Startup? File saving? Computation?
      - Better to analyse what aspects you want to be fast, and specify a time limit for them.

Characteristics of requirements

- Verifiable
  - Example requirements that can't be tested.
    - "The system must be able to store all of our customer's details."
      - This could be a moving target - their customer base may be increasing as you develop the product. You should have a fixed figure.
      - "The system should be able to initially store 50,000 customers, and be capable of expanding to 200,000 customers at a small cost".
      - Note you would also have to define what "a small cost" meant!

Characteristics of requirements

- Complete
  - The requirements should not leave out any areas.
  - This can be very hard to achieve, especially in large systems.
  - Current trends are to break the development up into smaller modules and implement each of them.
    - As you can test each implementation, you will have greater faith that requirements have not been overlooked.

Characteristics of requirements

- Non ambiguous
  - A requirement is ambiguous if it has more than one possible meaning.
    - "This dog will eat off your hand"
  - May be ambiguous due to
    - Poor choice of words
    - Differing definitions of a word or phrase.

Characteristics of requirements

- Ambiguous requirements
  - Poor choice of words
    - “Apple is working to improve the rest of the Mac OS code to allow for less busy loops from their side.”
    - Does this mean that there will be fewer busy loops, or the loops that do exist will be less busy?
    - We can see that the problem lies with the word “less” and the uncertainty as to which noun it applies.
      - less (busy loops)
      - (less busy) loops

Characteristics of requirements?

- Ambiguous requirements
  - Differing definitions of a word
  - This is especially likely when the common meaning of a word is different from that used in a specialised area.
    - "The system will store the time of each transaction"
      - Should this be the local time, or Universal time?
      - What precision should the time be recorded in?
    - In statistics "significant" has a specific meaning ("there is evidence of a systematic relationship") that is different from the general meaning ("having or expressing a meaning")

Characteristics of requirements

- Consistent
  - All parts of the requirements document should be consistent with each other.
  - They should not contradict each other (the following do).
    - "If you delete a task all of its subtasks should be deleted automatically"
    - "You should always be prompted to confirm the deletion of a task and any of its subtasks"
  - The larger the set of requirements the more likely it is that it will be inconsistent.

Characteristics of requirements

- Modifiable
  - The requirements should be structured so that it is possible to change it at little to no cost.
  - Requires it be structured carefully, e.g.
    - Separating functional from non functional requirements
    - Supplying a glossary (a table of definitions)
  - Use of appropriate tools that allow you to browse and automate modifications
    - Use of hyperlinked (HTML) documentation.
    - A tool similar to Sun's Javadoc would be ideal.

Characteristics of requirements

- Traceable
  - The requirements should be structured so that it is possible
    - To uniquely identify each requirement
      - Each one should have a unique number (e.g. "10.4.2")
      - This allows it to be referred to in final testing of the delivered software, and in discussions with a client.
    - To identify its history
      - Who requested the feature?
      - Is it the result of any other feature?
      - Who authorised it.
      - Dates decisions made

Requirements engineering

- How to get the requirements from the client
  - Gathering requirements can be very difficult
    - The client may not have a clear idea of what they want
    - The problem may be very large and take a long time to fully understand
    - The requirements may be rapidly changing
    - The client may not be fully committed to the project and will not spend the time with you that you need.
    - They may not understand what you need to complete the task.
    - They may have unreal expectations of what can be achieved

Requirements engineering

- Requirements engineering is…
  - requirements elicitation
  - requirements analysis and negotiation
  - requirements specification/documentation
  - requirements validation

Defining terms

- Requirements documents contain a lot of domain specific terms
- It is important to clearly define these terms so that people have a single point where they can clarify them
- These go into a Data Dictionary - a large table of definitions
- Write down a definition of “parent”

Definitions

- Does your definition consider...
  - Adopted children
  - Children who have had a number of parents through adoption
  - Children whose parents are the government (orphans)
- Is it biologically based or socially based?